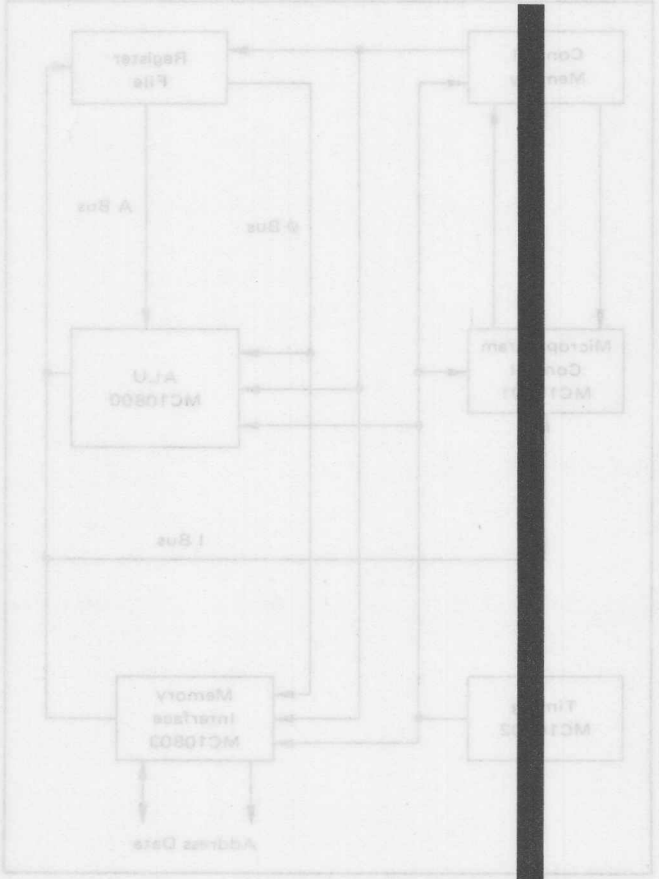
AN-776
Application Note

THE M10800 MECL LSI PROCESSOR FAMILY — THREE DESCRIPTIVE ARTICLES —



"GET THE BEST PROCESSOR PERFORMANCE"
By:
Tom Balph
LSI System Engineer
and
Bill Blood
Manager, Bipolar LSI System Engineering
Motorola I.C. Division, Mesa, Arizona

Reprinted from *Electronic Design*, June 7, 1977, Vol. 25, No. 12.

"THE MSP: GTE SYLVANIA'S MICROSIGNAL PROCESSOR"
By:
Howard I. Cohen
GTE Sylvania, Inc., Needham Heights, Massachusetts.

Reprinted from Proceedings of *Electro 77*, Session 38/3, April, 1977.

"M10800 MICROPROGRAMMED DEMONSTRATOR"
By:
Tom Balph
LSI System Engineer
and
Ian LeMair,
Field Applications Engineer
Motorola I.C. Division, Mesa, Arizona

Reprinted from Proceedings of *Electro 77*, Session 31/2, April 1977.

This note consists of three articles about the M10800 Processor Family. The first article examines the features of the processor components. The second describes a customer's application of M10800 devices and the third, an EXORciser-based M10800 microprogramming demonstrator.

MOTOROLA Semiconductor Products Inc.

Get the Best Processor Performance

by building it from ECL bit slices. The 10800 family offers versatility as well as cycle times of less than 100 ns.

Prepared by:
Tom Balph, LSI System Engineer
Bill Blood, Manager, Bipolar LSI System Engineering
Motorola I.C. Division
Mesa, Arizona

Offering the fastest cycle times of any available bit-slice processor family, the 10800 series of ECL 4-bit processor slices permits you to design high-speed computer systems. Each 10800 circuit is completely expandable, and the major system building blocks are either available or being designed.

The core of any 10800-based system, the arithmetic and logic unit (ALU), operates at system-clock frequencies of 10 to 15 MHz, which represent cycle times of 60 to 100 ns. System word size starts at the ALU width of 4 bits, but can be expanded to n x 4 bits by cascading ALU sections. To support the ALU, Motorola has developed several ECL circuits that take care of most of the housekeeping without restricting the processor design.

Intended to address the instructions stored in the microprogram memory, the 10801 microprogram controller provides a 4-bit address that can be expanded to any size by cascading controllers. A memory interface circuit, the 10803, also has a cascadable 4-bit output bus, but it connects to the address bus of the main memory and supplies all the read and write addresses.

Acting as a register file, stack or I/O buffer, the 10806 dual-port memory provides 32 words x 9 bits of temporary storage and can be accessed through either of its ports. For high-speed mathematical operations, the 10808 multibit shifter can handle up to 16 bits and, under software control, can do left-shift, right-shift or rotate operations. Additional 10808s can be cascaded for larger word lengths.

Other support circuits include the 10802 timing generator and clock controller, the 10804 and 10805 bidirectional bus translators (ECL to TTL and vice-versa) and all of the MECL 10,000 series of logic circuits. (All 10800 series circuits and some of the most commonly used 10,000-series units are listed in Table 1.)

A basic processor, built from 10800-series circuits, is outlined in Fig. 1. With a 16-bit word length, the

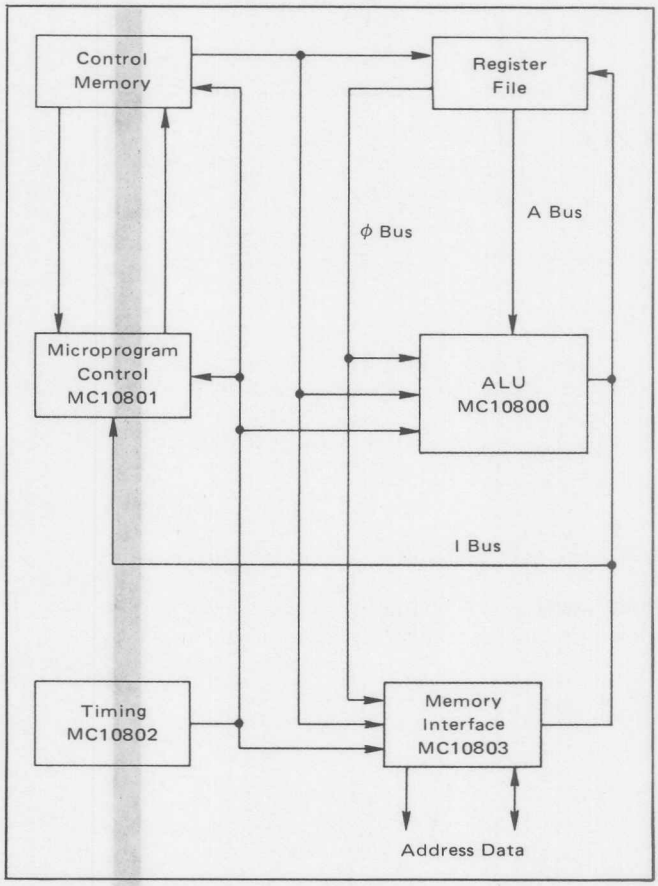


FIGURE 1 – With the 10800 series of bit slices, you can build a high-performance microprogrammable processor with instruction times of less than 100 ns.

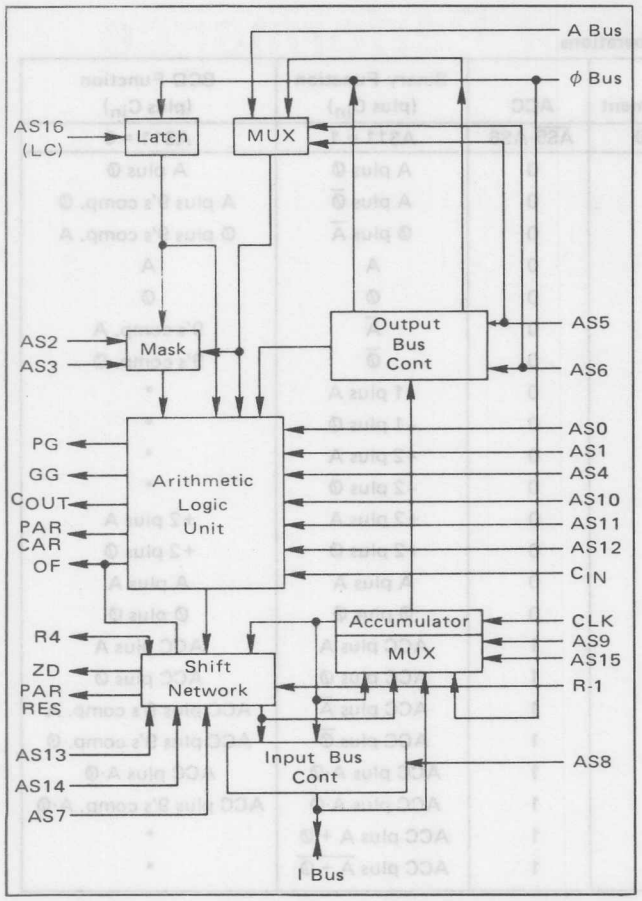


FIGURE 2 — Capable of 28 logic and 23 arithmetic operations as well as 16-data-routing options, the M10800 4-bit ALU slice can form the heart of a flexible computer system.

processor would typically consist of four 10800s, two or more 10801s, one 10802, four 10803s, one 10179 (look-ahead carry generator), several 10145s or 10143s for a register file, some microprogram and main memory, and of course some logic glue (SSI and MSI packages). But getting all the parts to work in unison can be quite tricky unless you know how they work.

TABLE 1
M10800 Components and Support

Part Number	Description
MC10800	Arithmetic and logic unit
MC10801	Microprogram sequencer
MC10802	Timing function and clock controller
MC10803	Memory-interface circuit
MC10804	4-Bit bidirectional bus translater (ECL/TTL)
MC10805	5-Bit bidirectional bus translator (ECL/TTL)
MC10806	Dual-port address register (32 x 9)
MC10808	Multibit shifter
MCM10143	Multiport RAM (8 x 2)
MCM10145	Single-port RAM (16 x 4)
MCM10146	Single-port RAM (1 k x 1)
MCM10149	ECL PROM (256 x 4)
MC10176	Hex, master-slave flip-flop
MC10179	Lookahead carry generator

GET TO KNOW THE ALU

Start with the core. The 10800 4-bit ALU slice can perform both binary and BCD arithmetic and logic operations on combinations of one, two, or three variables. All 10800 operations are too numerous to list but some commonly used commands include 28 logic, 23 arithmetic and 16 data-routing combinations as shown in Tables 2, 3, and 4, respectively. Housed in a 48-pin quad-in-line package (QUIL), the ALU slice offers a flexible organization (Fig. 2).

There are three 4-bit ports on the 10800: an input-only data bus (A bus) and two bidirectional buses (Ø and I buses). The A and Ø buses are the primary data-entry ports, and the I bus is the main output bus. Accepting data from the A bus, Ø bus and/or the internal accumulator, the ALU performs its operations under the direction of 17 selection and control lines.

The three buses and select lines account for 29 pins on the 10800. Another eight pins are required for power (four for grounds, two for -2 V and two for -5.2 V). Five additional lines provide ALU-status outputs, another line is for the clock input, one more for the carry input, and two for parity outputs (one for the carry parity and

TABLE 2
Basic ALU Logic Commands

Y MUX		X MUX		INV	ACC	Function
AS0	AS1	AS2	AS3	AS10	$\overline{AS5 \cdot AS6}$	
0	1	0	1	1	0	Logic 0
0	0	1	0	1	0	A
0	0	0	1	1	0	Ø
0	0	1	0	0	0	$\overline{A}$
0	0	0	1	0	0	$\overline{Ø}$
0	0	1	1	1	0	A + Ø
0	1	0	0	0	0	A + $\overline{Ø}$
1	0	0	0	0	0	$\overline{A}$ + Ø
0	0	0	0	1	0	A · Ø
0	1	1	1	1	0	A · $\overline{Ø}$
0	1	0	0	1	0	$\overline{A}$ · Ø
0	1	1	0	1	0	A ⊕ Ø
0	1	1	0	0	0	$\overline{A \oplus Ø}$
0	0	0	0	0	0	$\overline{A \cdot Ø}$
0	0	1	1	0	0	$\overline{A + Ø}$
0	1	0	1	0	0	Logic 1
1	0	1	0	1	1	ACC · $\overline{A}$
0	1	0	1	1	1	ACC · $\overline{Ø}$
1	0	1	0	0	1	$\overline{ACC}$ + A
0	1	0	1	0	1	$\overline{ACC}$ + Ø
0	0	1	0	1	1	ACC ⊕ A
0	0	1	0	0	1	ACC ⊕ $\overline{A}$
0	0	0	1	1	1	ACC ⊕ Ø
0	0	0	1	0	1	ACC ⊕ $\overline{Ø}$
0	0	0	0	1	1	ACC ⊕ A · Ø
0	0	0	0	0	1	ACC ⊕ $\overline{A \cdot Ø}$
0	0	1	1	1	1	ACC ⊕ A + Ø
0	0	1	1	0	1	ACC ⊕ $\overline{A + Ø}$

TABLE 3
ALU Arithmetic Operations

Y MUX		X MUX		±2	Complement	ACC	Binary Function (plus C_{in})	BCD Function (plus C_{in})
AS0	AS1	AS2	AS3	AS4	AS10	$\overline{AS5}$·AS6	AS11 = 1	AS11 = 0
1	0	0	1	1	1	0	A plus Ø	A plus Ø
1	0	0	1	1	0	0	A plus $\overline{Ø}$	A plus 9's comp. Ø
0	1	1	0	1	0	0	Ø plus $\overline{A}$	Ø plus 9's comp. A
0	0	1	0	1	1	0	A	A
0	0	0	1	1	1	0	Ø	Ø
0	0	1	0	1	0	0	$\overline{A}$	9's comp. A
0	0	0	1	1	0	0	$\overline{Ø}$	9's comp. Ø
1	1	1	0	1	1	0	-1 plus A	*
1	1	0	1	1	1	0	-1 plus Ø	*
1	1	1	0	0	1	0	-2 plus A	*
1	1	0	1	0	1	0	-2 plus Ø	*
0	0	1	0	0	1	0	+2 plus A	+2 plus A
0	0	0	1	0	1	0	+2 plus Ø	+2 plus Ø
1	0	1	0	1	1	0	A plus A	A plus A
0	1	0	1	1	1	0	Ø plus Ø	Ø plus Ø
0	0	1	0	1	1	1	ACC plus A	ACC plus A
0	0	0	1	1	1	1	ACC plus Ø	ACC plus Ø
0	0	1	0	1	0	1	ACC plus $\overline{A}$	ACC plus 9's comp. A
0	0	0	1	1	0	1	ACC plus $\overline{Ø}$	ACC plus 9's comp. Ø
0	0	0	0	1	1	1	ACC plus A·Ø	ACC plus A·Ø
0	0	0	0	1	0	1	ACC plus $\overline{A·Ø}$	ACC plus 9's comp. A·Ø
0	0	1	1	1	1	1	ACC plus A + Ø	*
0	0	1	1	1	0	1	ACC plus $\overline{A + Ø}$	*

*Not defined in BCD

one for the result parity). And two more control the shift-left/shift-right operation of the ALU's output-shift register. Shifting is possible for the A or Ø bus as well as the accumulator. Also various add or subtract/shift combinations can be performed by using the shift network with the ALU.

TABLE 4
Data-Transfer Paths in ALU

				Function		
AS7	AS8	AS9	AS15	ACC Source	Shift Source	Input Bus
0	0	0	0	RES	ACC	Disable
0	0	0	1	ØB	ACC	Disable
0	0	1	0	IB	ACC	Disable
0	0	1	1	ACC	ACC	Disable
0	1	0	0	RES	ACC	ACC
0	1	0	0	ØB	ACC	RES
0	1	1	0	IB	ACC	RES
0	1	1	1	ACC	ACC	RES
1	0	0	0	RES	F_{out}	Disable
1	0	0	1	ØB	F_{out}	Disable
1	0	1	0	IB	F_{out}	Disable
1	0	1	1	ACC	F_{out}	Disable
1	1	0	0	RES	F_{out}	ACC
1	1	0	1	ØB	F_{out}	RES
1	1	1	0	IB	F_{out}	RES
1	1	1	1	ACC	F_{out}	RES

Whether the ALU does binary or BCD arithmetic, the speed of the operation remains the same: A 9's complement circuit is built in and automatically enabled whenever BCD operations are performed. In addition, only one line, AS11, must change state to switch operating modes (see Table 2).

A masking multiplexer on the chip precedes the ALU and performs bit manipulations on incoming data. An on-chip accumulator can hold data for ALU operations. The I and Ø ports are connected to both the ALU and accumulator thus increasing data processing flexibility. With the accumulator, A bus, and Ø bus as operands, any of the functions listed in Tables 2, 3, or 4 can be performed in one pass through the circuit.

The status-output lines from the 10800 include the carry-propagate and carry-generate signals for look-ahead carry operations. A carry-output signal is available when slices are cascaded, without carry look-ahead. And also available are an overflow output used only for binary operations for indicating when the maximum output from the ALU has been exceeded, and a zero-detect line for indicating when an all-zero condition exists in the output of the ALU shift network. The overflow output can also be used to detect sign changes that stem from a shift operation.

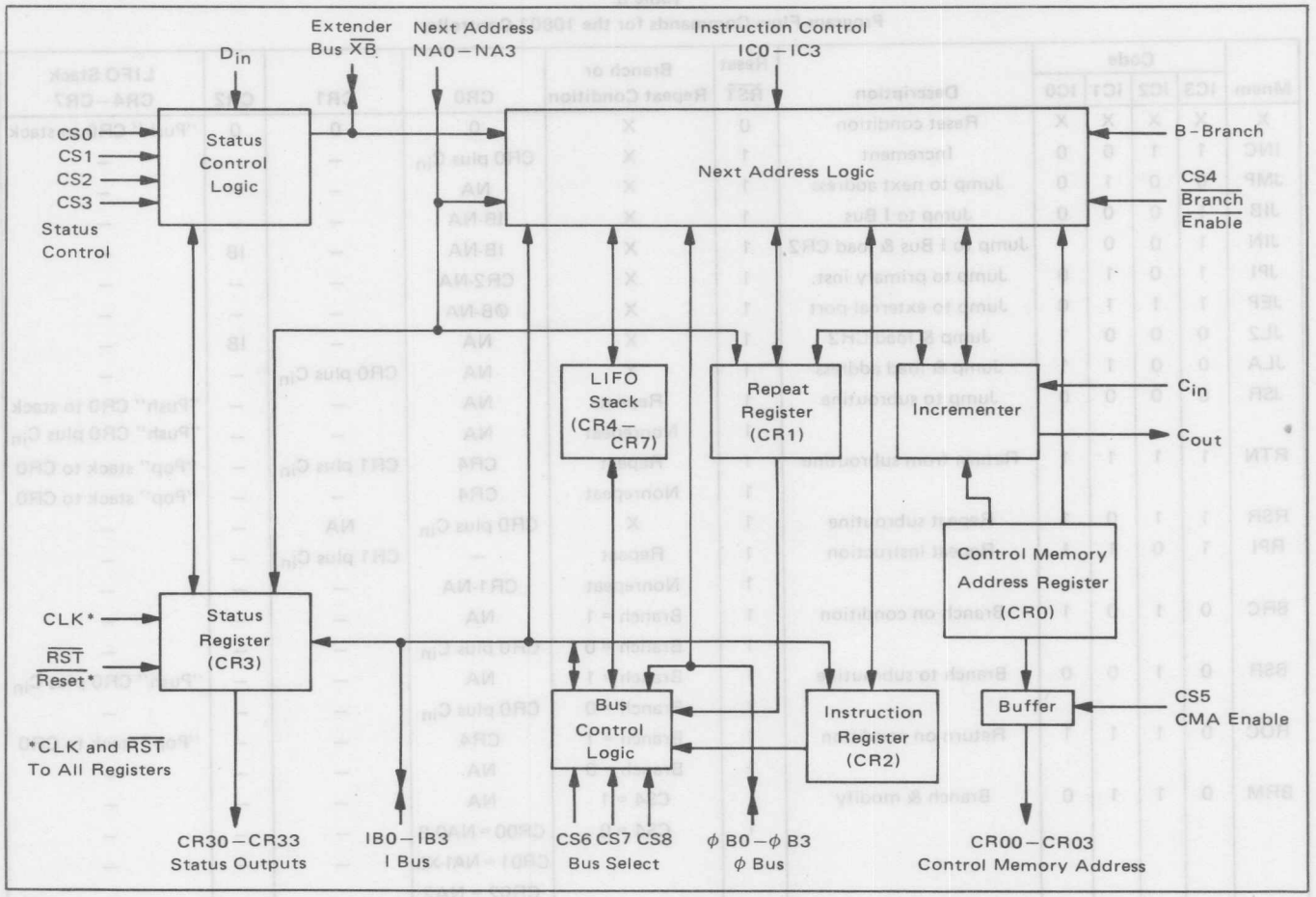


FIGURE 3 – Controlling the microprogram addressing, the M10801 microprogram sequencer generates the next address and has a repertoire of 16 program-flow commands. It provides a 4-bit section of the address.

MANIPULATE THE ALU WITH THE MICROPROGRAM

When a processor is built from 10800 ALU slices, a microprogram selects an ALU operation every clock cycle. The 10801 controller (Fig. 3) determines the next microprogram memory location to be addressed. It performs incremental sequencing, handles jumps, conditional branches, subroutines and repeat loops.

Inside the chip, the control-memory-address register (CR_0) holds the microprogram address that accesses the next instruction. A next-address logic block, controlled by various signals, selects the microprogram's next instruction address and routes it to the CR_0 inputs.

The 10801 has 16 instructions to control the program flow (Table 5). These commands include incremental sequencing and several types of jump address operation. Sources of jump addresses include the I and $\emptyset$ bus, the next address inputs, CR_1 and CR_2.

Conditional operations include a branch on condition command that looks at the branch input line B, the branch expansion input $\overline{XB}$ or status bits in the CR_3 register and makes a program flow decision. Another instruction, the BRM (branch and modify) command, uses the branch status input, B, and the $\overline{XB}$ line to modify the next microprogram address for a four-way branch in the program.

Subroutines can operate in either a repeat or nonrepeat mode. The nonrepeat operation is simply a jump to subroutine followed by a return to subroutine. However in the repeat mode, the current subroutine is automatically repeated a specific number of times, as determined by the CR_1 register contents.

Other registers inside the 10801 contribute to the microprogram sequencing. The CR_1 register can be used either as a cycle counter for repeat sequences or to store a microprogram address when servicing interrupts. Register CR_2 can hold a machine-instruction starting address or an interrupt vector for microprogram flow jumps.

The CR_3 register stores the status bits for conditional program jumps. These bits are most often external signal lines, such as interrupts or test points.

Sometimes, the ALU's condition-code bits are held in CR_3, but usually they are stored in a separate register controlled by the microprogram. Select lines CS_0, CS_1, CS_2, and CS_3 control the CR_3 status register and permit it to store or read individual bits. All status conditions are available as outputs.

Another bank of four registers, CR_4 to CR_7, forms a last-in, first-out (LIFO) stack to next subroutines. Controlled by the next-address logic block, the stack can be extended through the $\emptyset$ port bus with external memory. Also, should an emergency power-down or priority interrupt be signaled, all the register's contents can be

Table 5
Program-Flow Commands for the 10801 Controller

Mnem	Code IC3	IC2	IC1	IC0	Description	Reset $\overline{RST}$	Branch or Repeat Condition	CR0	CR1	CR2	LIFO Stack CR4–CR7
X	X	X	X	X	Reset condition	0	X	0	0	0	"Push" CR0 to stack
INC	1	1	0	0	Increment	1	X	CR0 plus C_{in}	–	–	–
JMP	0	0	1	0	Jump to next address	1	X	NA	–	–	–
JIB	1	0	0	0	Jump to I Bus	1	X	IB-NA	–	–	–
JIN	1	0	0	1	Jump to I Bus & load CR2	1	X	IB-NA	–	IB	–
JPI	1	0	1	0	Jump to primary inst.	1	X	CR2-NA	–	–	–
JEP	1	1	1	0	Jump to external port	1	X	ØB-NA	–	–	–
JL2	0	0	0	1	Jump & load CR2	1	X	NA	–	IB	–
JLA	0	0	1	1	Jump & load address	1	X	NA	CR0 plus C_{in}	–	–
JSR	0	0	0	0	Jump to subroutine	1	Repeat	NA	–	–	"Push" CR0 to stack
						1	Nonrepeat	NA	–	–	"Push" CR0 plus C_{in}
RTN	1	1	1	1	Return from subroutine	1	Repeat	CR4	CR1 plus C_{in}	–	"Pop" stack to CR0
						1	Nonrepeat	CR4	–	–	"Pop" stack to CR0
RSR	1	1	0	1	Repeat subroutine	1	X	CR0 plus C_{in}	NA	–	–
RPI	1	0	1	1	Repeat instruction	1	Repeat	–	CR1 plus C_{in}	–	–
						1	Nonrepeat	CR1-NA	–	–	–
BRC	0	1	0	1	Branch on condition	1	Branch = 1	NA	–	–	–
						1	Branch = 0	CR0 plus C_{in}	–	–	–
BSR	0	1	0	0	Branch to subroutine	1	Branch = 1	NA	–	–	"Push" CR0 plus C_{in}
						1	Branch = 0	CR0 plus C_{in}	–	–	–
ROC	0	1	1	1	Return on condition	1	Branch = 1	CR4	–	–	"Pop" stack to CR0
						1	Branch = 0	NA	–	–	–
BRM	0	1	1	0	Branch & modify	1	CS4 = 1	NA	–	–	–
						1	CS4 = 0	CR00 = NA0-B	–	–	–
								CR01 = NA1-XB	–	–	–
								CR02 = NA2			
								CR03 = NA3			

"dumped" through the Ø or I bus ports. The reverse data flow is also possible–all registers can be loaded through the same ports.

The 10801 provides a 4-bit section of the microprogram address, and any number of controllers can be cascasded. For instance, two 10801s can control 16 pages of 256 words. One of the available CR_3 registers makes a handy page-address register while the other available CR_3 register can be used to hold status information. Larger systems might use three 10801s to address up to 4096 words directly or 16 pages of 4096 words each.

The 10801 has five I/O ports–the CR_0, CR_3, I bus, Ø bus and NA inputs–which account for 20 of the 48 pins on the QUIL package. Another nine pins are used for the select lines, four more are the instruction inputs, and eight are needed for power and ground. Two more pins form the carry-in and carry-out lines for cascading, one is for the clock input and four take care of other control functions.

MAIN MEMORY ALSO NEEDS A CONTROLLER

To generate main memory addresses for any 10800-based system, the 10803 I/O controller interfaces to the main memory and peripherals via the address and data buses. I and Ø bus ports interface to the other system components. A memory-data register inside the 10803 chip (Fig. 4) holds incoming or outgoing data, and a memory-address register (MAR) holds outgoing information. A 4 x 4 register file can be used as a program counter, a stack pointer, an index register or other related functions. Its output feeds into the simple ALU or into the multiplexer input to the memory-address register.

A data-matrix block controls the transfer of data between various internal registers and I/O ports. Seventeen data-transfer operations are possible (Table 6a and b), and they are controlled by inputs MS_0 to MS_3, MS_5 and MS_{14}. The MS_{14} line permits any combination of positive and negative logic formats by inverting any data on the data and address buses. Once selected, any transfer can be performed within one microinstruction.

The 10803's ALU performs AND, OR, Exclusive-OR, add, subtract, shift-left and shift-right operations to compute the extended, indexed and relative addresses and perform stack-pointer operations. Controlled by the microfunction and destination-decode logic, the ALU gets operands from the Ø or I bus, the MDR, the register file, the program counter, the MAR or the P inputs. The four P-input lines are pointer inputs that can be used to add address offsets, increment or decrement the address, or supply mask bits. And since independent select lines manipulate the ALU and data matrix, the 10803 can perform two independent parallel operations during one microinstruction cycle.

Housed in the same 48-pin QUIL case as the 10800 and 10801, the 10803 has its pins allocated as follows: four 4-pin buses for memory data, memory address,

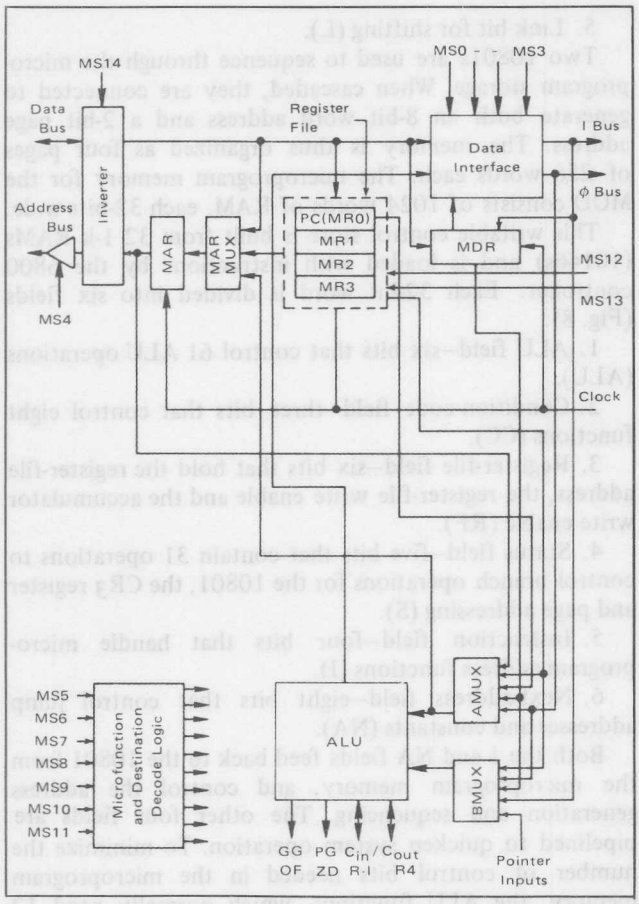


FIGURE 4 — Built with five I/O ports, the M10803 memory interface circuit offers very flexible operation along with 17 different data-routing instructions.

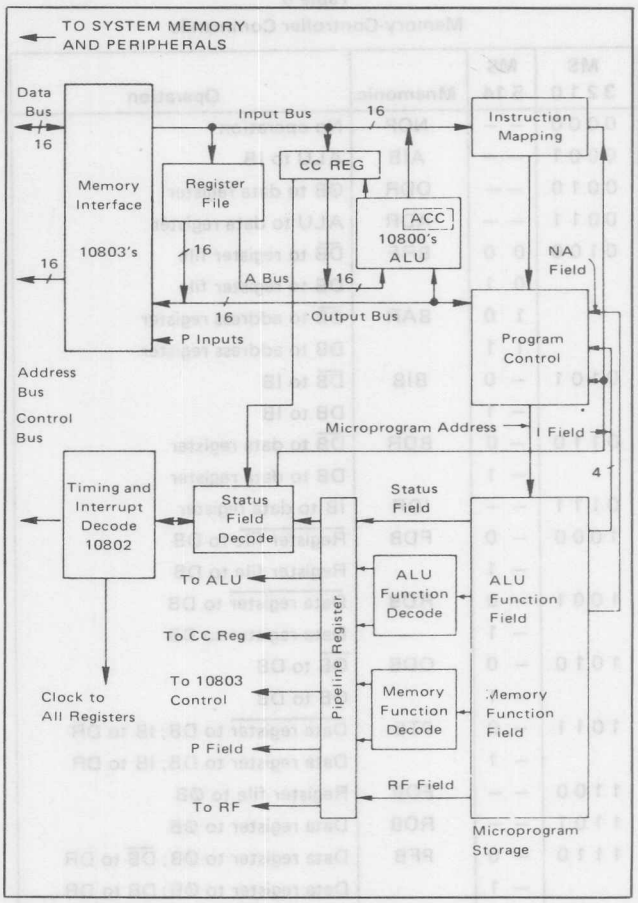


FIGURE 5 — A generalized 16-bit minicomputer can be based on the 10800 bit slices. Its pipelined architecture offers the best performance technology can provide.

input and output, four more pins for pointer inputs, 11 more for data and address-select lines, another eight for power, one for output enable, one for a clock, two for carry lines, two for register-file select, two for carry propagate and generate, and one for data and address inversion.

To get all the blocks to work in unison, the 10802 timing function circuit generates up to four clock phases from a single-phase oscillator input. It also allows for system starts and stops, and has a single-step control for diagnostic purposes. Both the number of clock phases and their duration are programmable through seven select lines.

The 10802 is housed in a 24-pin DIP with seven pins pins for the select inputs, four for the phase outputs, three for power, one for the clock input and nine for controls. Several timing circuits can be cascaded if more than four clock phases are needed. The 10802 can also be programmed to compensate for one slow path without slowing down the faster sections by doubling the duration of any clock phase.

SYNCHRONIZE ALL THE LOGIC BLOCKS

At this point, getting all the parts to work together is a matter deciding on word length, system architecture and microprogramming structure. A generalized 16-bit minicomputer architecture, based on the 10800 series, is shown in Fig. 5. The system uses a pipelined architecture to keep the cycle time as fast as possible. The actual fetch and execution of a microinstruction are accomplished in two clock cycles. The next instruction is being fetched while the current instruction is being executed.

A microprogram control cycle consists of addressing the microprogram memory, setting up a new address (through the I and NA fields) and clocking the new address to the 10801's CR_0 register, which starts the cycle all over again. During the control cycle, new data-processing and memory interface operations are accessed and clocked into the pipeline register. A status-field, also held in the pipeline register, is used to select branch conditions, update interrupt status, and reset the system, among other functions. The number of microprogram memory bits is minimized by decoding the various fields from the microprogram memory.

Before setting up a microcode to control all these operations, a system architecture and instructions set must be defined. Look at a master-slave system that uses a 6800 microprocessor as a controller and an 8-bit, 10800-based processor to rapidly perform complex mathematical operations (Fig. 6). For more about the 6800, see "Microprocessor Basics: Part 5," ED No. 15, July 19, 1976, p. 66. The arithmetic subsystem, called the MOD processor, can be built on three EXORciser-compatible boards.

The 6800 acts as the master controller for the MOD,

Table 6
Memory-Controller Commands

MS 3 2 1 0	MS 5 14	Mnemonic	Operation
0 0 0 0	– –	NOP	No operation
0 0 0 1	– –	AIB	ALU to IB
0 0 1 0	– –	ODR	ØB to data register
0 0 1 1	– –	ADR	ALU to data register
0 1 0 0	0 0	BRF	$\overline{DB}$ to register file
	0 1		DB to register file
	1 0	BAR	$\overline{DB}$ to address register
	1 1		DB to address register
0 1 0 1	– 0	BIB	$\overline{DB}$ to IB
	– 1		DB to IB
0 1 1 0	– 0	BDR	$\overline{DB}$ to data register
	– 1		DB to data register
0 1 1 1	– –	IDR	IB to data register
1 0 0 0	– 0	FDB	$\overline{\text{Register file}}$ to DB
	– 1		Register file to DB
1 0 0 1	– 0	RDB	$\overline{\text{Data register}}$ to DB
	– 1		Data register to DB
1 0 1 0	– 0	ODB	$\overline{ØB}$ to DB
	– 1		ØB to DB
1 0 1 1	– 0	PTB	$\overline{\text{Data register}}$ to DB; IB to DR
	– 1		Data register to DB; IB to DR
1 1 0 0	– –	FOB	Register file to ØB
1 1 0 1	– –	ROB	Data register to ØB
1 1 1 0	– 0	PFB	Data register to ØB; $\overline{DB}$ to DR
	– 1		Data register to ØB; DB to DR
1 1 1 1	– –	MOR	Data register to ØB; IB to DR

Operands	Functions	Destinations
Ø bus	A plus B	I bus
I bus	A plus $\overline{B}$	Address register
Address register	A · B, A · P	Data Register
Data register	A + B	Register file
Register file	A ⊕ B, A ⊕ P	Program counter
Program counter	A plus A	
Pointer inputs	Shift right	
	A plus P	
	PC plus B	
	AR plus B	

and although the MOD runs independently when number crunching, the 6800 oversees the data transfer, and loads and modifies the microprogram storage. The actual MOD number-crunching section consists of two 10800 ALUs and 16 register-file locations made from the 10145 ECL RAMs (Fig. 7). Data entering the MOD are routed through the ALU and into the register file. The accumulator, register file and condition-code register are loaded directly from the outgoing data bus.

Five bits contained in the condition-code register are used to indicate the following:

1. Carry output from the most-significant bit of the ALU (C).
2. Two's complement arithmetic overflow (V).
3. Zero detect output of the result (Z).
4. Sign of the result (N).
5. Link bit for shifting (L).

Two 10801s are used to sequence through the microprogram storage. When cascaded, they are connected to generate both an 8-bit word address and a 2-bit page address. The memory is thus organized as four pages of 256 words each. The microprogram memory for the MOD consists of 1024 words of RAM, each 32-bits wide.

This writable control store is built from 32 1-k RAMs (10146s) and is loaded with instructions by the 6800 controller. Each 32-bit word is divided into six fields (Fig. 8):

1. ALU field–six bits that control 61 ALU operations (ALU).
2. Condition-code field–three bits that control eight functions (CC).
3. Register-file field–six bits that hold the register-file address, the register-file write enable and the accumulator write enable (RF).
4. Status field–five bits that contain 31 operations to control branch operations for the 10801, the CR_3 register and page addressing (S).
5. Instruction field–four bits that handle microprogram address functions (I).
6. Next-address field–eight bits that control jump addresses and constants (NA).

Both the I and NA fields feed back to the 10801 from the microprogram memory, and control the address generation and sequencing. The other four fields are pipelined to quicken system operation. To minimize the number of control bits needed in the microprogram memory, the ALU functions, which normally need 12 lines, are encoded into three 10149 PROMs, which are, in turn, fed by the six bits of the ALU field. Located before the pipeline register, the decoding PROMs do not reduce system speed.

The system software for the 6800/10800 combination consists of four parts written in 6800 code:

1. System initialization and writable control-store monitor to interface with the control storage.
2. A punch or memory dump to store the control memory contents in cassettes.
3. A memory load from cassette.
4. A monitor to control, load and receive data from the processor.

LOAD THE CODE IN BYTES

Information from the 6800 is written into the writable control store in 1-byte chunks, with four individual writes from the 6800 needed for each microword. After all the operations you want the 10800 to perform have been defined, your next task is to write the microprogram. If programs are to be written by hand, a tabular listing, as shown in Table 7, can simplify the procedures.

Line (1) represents a complete microinstruction in mnemonic form. Each of the field columns defines an operation performed by the microword: INC–increment address; X–don't care for next address; INHS–inhibit status; BADA–binary-add register field and accumulator; LAR–load Z, N, C and V condition-code bits; RFB–register B; FE = 0–disable register-file write; and AE = 1–enable accumulator write. In "English" the microword tells the machine to increment the address, add register B

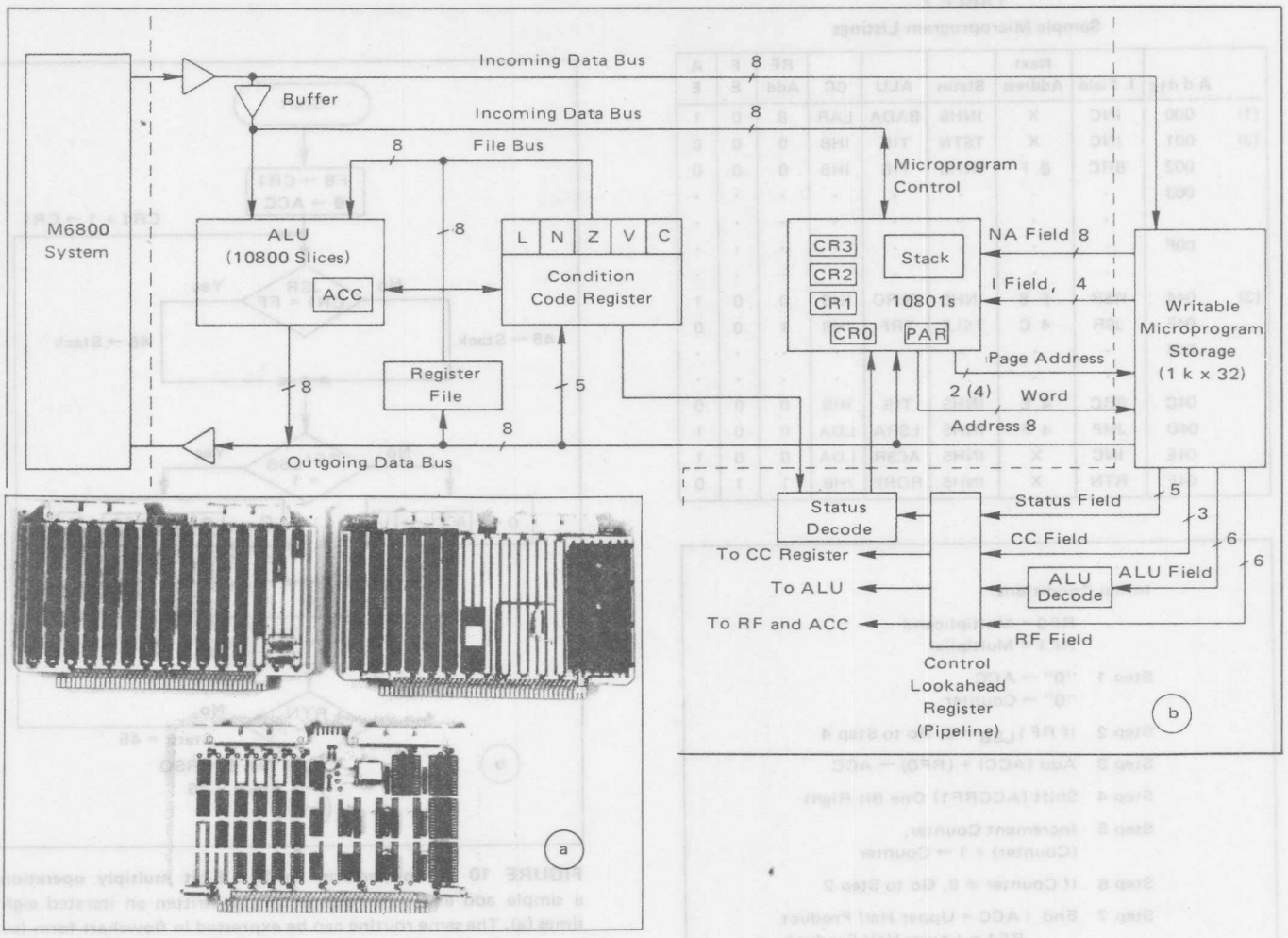


FIGURE 6 — With a 6800 μP acting as controller, a dedicated arithmetic processor can be built from 10800-series components (a). Multiply operations can be done in less than 1/100 the time needed by the 6800 alone. Only three EXORciser-compatible boards are needed to build the slave processor (b).

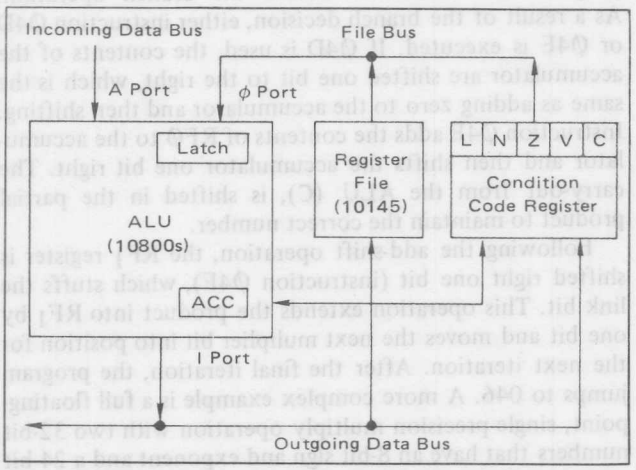


FIGURE 7 — The data-processing section of the MOD boards consists of an 8-bit ALU, a 16 x 8 register file and a condition-code register.

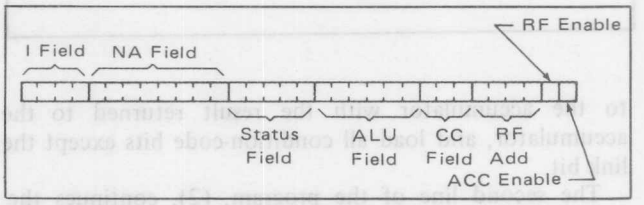


FIGURE 8 — The microinstruction that controls MOD operation is subdivided into six fields that control smaller sections of the processor.

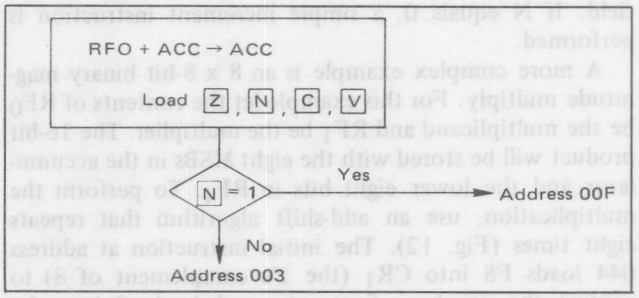


FIGURE 9 — Describing the operation defined in Table 7's line (2), this simple flow chart shows how the sequencer can perform a branch operation.

TABLE 7
Sample Microprogram Listings

	Add_{16}	I. Field	Next Address	Status	ALU	CC	RF Add	F E	A E
(1)	000	INC	X	INHS	BADA	LAR	B	0	1
(2)	001	INC	X	TSTN	TIB	IHB	0	0	0
	002	BRC	Ø F	INHS	TIB	IHB	0	0	0
	003	.	.	.	.	.	.	.	.
		.	.	.	.	.	.	.	.
	00F	.	.	.	.	.	.	.	.
		.	.	.	.	.	.	.	.
(3)	044	RSR	F 8	INHS	ZERO	IHB	0	0	1
	045	JSR	4 C	TSLS	TRF	IHB	1	0	0
	046	.	.	.	.	.	.	.	.
		.	.	.	.	.	.	.	.
	04C	BRC	4 E	INHS	TIB	IHB	0	0	0
	04D	JMP	4 F	INHS	LSRA	LDA	0	0	1
	04E	INC	X	INHS	ACSR	LDA	0	0	1
	04F	RTN	X	INHS	RORF	IHB	1	1	0

Initial Conditions:

RF0 = Multiplicand
RF1 = Multiplier

Step 1 "0" → ACC
"0" → Counter

Step 2 If $RF1_{LSB}$ = 0, Go to Step 4

Step 3 Add (ACC) + (RF0) → ACC

Step 4 Shift (ACCRF1) One Bit Right

Step 5 Increment Counter,
(Counter) + 1 → Counter

Step 6 If Counter ≠ 8, Go to Step 2

Step 7 End | ACC = Upper Half Product
RF1 = Lower Half Product

(a)

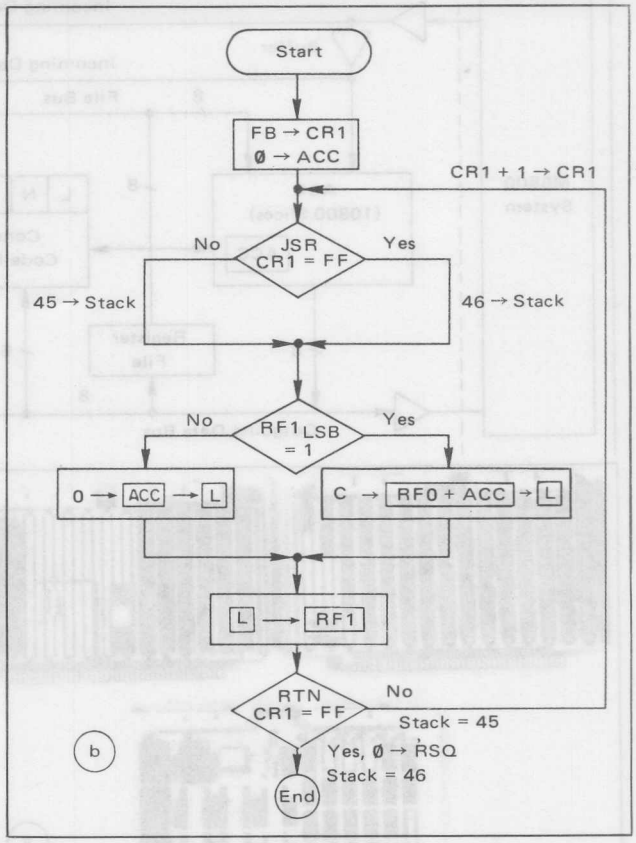


FIGURE 10 — To perform an 8 x 8-bit multiply operation, a simple add and shift routine can be written an iterated eight times (a). The same routine can be expressed in flowchart form (b).

to the accumulator with the result returned to the accumulator, and load all condition-code bits except the link bit.

The second line of the program, (2), continues the instruction at address 000 to the instructions at addresses 001 and 002. A register-file-add-to-accumulator instruction given in address 000 is followed by a branch operation at address 002 (Fig. 9). The I-field address advances to 002. If N equals 1 the branch occurs, and advances the address to ØØF, as specified by the NA field. If N equals 0, a simple increment instruction is performed.

A more complex example is an 8 x 8-bit binary magnitude multiply. For this example let the contents of RF_0 be the multiplicand and RF_1 be the multiplier. The 16-bit product will be stored with the eight MSBs in the accumulator and the lower eight bits in RF_1. To perform the multiplication, use an add-shift algorithm that repeats eight times (Fig. 12). The initial instruction at address 044 loads F8 into CR_1 (the 2's complement of 8) to control the number of repeats, and loads 0 into the accumulator. The repeat loop is started at address 045 with the JSR command.

Statement 045 sets up the test condition for the BRC at statement Ø4C. The ALU and status operations combine to put the contents of RF_1 on the output-data bus and test the LSB (RF_1 → ØB, Branch = LSB). Also, because of pipelining, this operation occurs in the microprogram one statement before the branch operation. As a result of the branch decision, either instruction Ø4D or Ø4E is executed. If Ø4D is used, the contents of the accumulator are shifted one bit to the right, which is the same as adding zero to the accumulator and then shifting. Instruction Ø4E adds the contents of RFØ to the accumulator and then shifts the accumulator one bit right. The carry-out from the ALU, (C), is shifted in the partial product to maintain the correct number.

Following the add-shift operation, the RF_1 register is shifted right one bit (instruction Ø4F), which stuffs the link bit. This operation extends the product into RF_1 by one bit and moves the next muliplier bit into position for the next iteration. After the final iteration, the program jumps to 046. A more complex example is a full floating-point, single-precision multiply operation with two 32-bit numbers that have an 8-bit sign and exponent and a 24-bit mantissa. With a 10 MHz clock the MOD processor needs 37.6 μs to perform the operation—less than 1/100 the time of an equivalent software multiply in the 6800.

THE MSP: GTE Sylvania's Microsignal Processor

Prepared by:
Howard I. Cohen
GTE Sylvania Incorporated, Eastern Division
77 "A" Street
Needham Heights, Massachusetts 02194

SUMMARY

Over a two-year period, GTE Sylvania examined the attributes of several bipolar bit-slice devices including the AMD2901, MMI6701, Intel 3001, and the MOT 10800, against the criteria of maximizing system speed in a real-time computational application and minimizing manufacturing production costs.

The 10800 was chosen because of its ready adaptability to a multi-bus architecture, higher speed and powerful family of support components. To ensure maximum flexibility in many different applications, a macro-code rather than a micro-code system was configured. It was given an effective three-address architecture and a Register File with 32 general purpose registers. Dual ALUs, one for index processing and one for general arithmetic operations permit simultaneity of actions of effective address calculation, incrementing or decrementing of index registers to manage data address selection, and arithmetic functions. Computational power is provided by the three-address (3-bus) structure which can be used with a REPEAT instruction to permit treating blocks of memory as arrays. Thus, in two lines of machine code (REPEAT, ACCUMULATE MULTIPLY) an entire auto-correlation calculation can be performed.

Control flexibility is ensured by mutli-function instructions such as REPEAT-COMPARE, a BIT TEST and BIT ALTER capability over every bit in memory. The separation of Program Memory and Data Memory ensures both speed of execution and ease of PROM implementation of dedicated applications. A very versatile I/O structure using memory managed I/O components permits easy adapting of I/O to a particular application.

Future systems will use more of the planned 10800 family products to improve performance and lower manufacturing costs.

PREFACE

Two different areas are covered in this paper. One reviews the factors that influenced technical decisions in developing a very powerful computer using elements of Motorola's Emitter Coupled Logic new LSI family called the 10800 series[1]. The other is the system itself and how the system configuration exploits the features of the 10800 family.

This paper assumes that the reader is conversant with the ISP (Instruction Set Processor) view of a computer, and more specifically with the published specifications and features of elements of the 10800 family. No attempt is made here to describe the elements as components, but rather how we have used them.

BACKGROUND

Over two years ago, GTE Sylvania's Eastern Division undertook an examination of the suitability of bit-slice LSI microprocessor devices for use in a family of advanced programmable signal processing systems. Its present family, called the PSP (Programmable Signal Processor)[2] has been used for several years in Linear Predictor Encoder,[3,4] Modem[5] and general filter applications interfaced to a variety of communication circuits. Its nominal performance characteristics are in the order of 2 to 4 MOPS (Million Operations Per Second) and performance approximately twice this capability was the goal for a new system, presumably using LSI components.

Devices were studied both in the lab and in possible system configurations including the Intel 3001, AMD2901, MMI6701 and the Motorola 10800 family. Detailed configuration designs were begun about a year ago. The 10800 was chosen over the others for three key reasons:

1. Its inherent speed, typical of the general competence of the ECL 10K family to support very fast, complex logical structures and drive 50 ohm distributed lines, is also capable of forming a full 16-bit sum, measured at under 40 nanoseconds.

2. The inherent architecture of the 10800 which is a three-port device, permitting it to be configured into a three-address, i.e., source-source-destination structure.

3. The family of ECL-LSI support devices, both available now and planned for this calendar year from Motorola. These include elements destined to enhance system performance as well as reduce system parts count and manufacturing costs.

This paper will describe the general structure of GTE Sylvania's MSP (Micro Signal Processor), a prototype of which has been fabricated and is undergoing application benchmark testing in our laboratories in Needham, Massachusetts. Emphasis will be given to exploitation of the 10800 family in order to realize operation in the 4 to 8 MOPS range.

SYSTEM ORGANIZATION

A decision based on a tradeoff study was made very early in the effort to implement a "classical-mini" rather than building a "wide-word" microprogram system, even though they are becoming more popular with the use of the bit-slice devices. This decision arose from the belief that ease of software function both in off-line development support and application was very important. Additionally, there was a very strong desire to salvage a large body of operational PSP code, at least at the symbolic language level. These strategies were tempered by the decision that the new MSP must be like, but not the same as the old PSP. This decision ensured that the architecture could be optimized to the devices, both present and future.

The computer system contains a full three bidirectional bus structure, uses a 16-bit data word and a separate 32-bit instruction word, not unlike a microprogrammed system, but with an instruction code format and organization characteristic of a macro-level ISP. The physical

division of data and program memory has two advantages: it permits independent access to improve operating speed, and easy expanding or contracting of the size of either data or instruction word independently of each other.

Other important overall features include an ISP heavily inclined toward arithmetic and decision-making functions. The emphasis in the I/O area is on control and versatility rather than speed or high volume data handling. The I/O was made versatile in order to accommodate very different application interfaces.

The busing structure exploits the inherent three-port capability of the 10800 family. The explicit functions of the three buses were carefully adapted to exploit the capabilities built into the 10800 and 10803. Not every desirable feature was found in these two key components, nor was every available feature used. However, the prevailing assessment is that these components have been conceived for relatively specialized functions and they became the keystone of our design. After many attempts at alternative configurations, the one shown in Figure 1 was adopted. The "source" buses are called the OPERAND bus and the DATA bus. The destination or RESULT bus inputs results back to memory. The system uses four 10800s for the Arithmetic Subsystem, and four 10803s for an Index Processor.

EXTENDED REGISTERS

All registers that are meaningful to the programmer are addressable and are called XR (extended registers). These include the ACCUMULATOR in the 10800 which functions as the Arithmetic/Logic element. The XRs also include the four file registers on-board the 10803, the first of which serves as a STACK POINTER register, while the last three are used as SIRs (Super Index Registers) which in turn support and index three address type operations. Note that all four registers are vitally involved in calculating or managing data memory addresses. Thus, the key parts of the 10803 including the on-board ALU and MAR are pressed into service to perform a memory interface function, namely, index processing on the one hand and subroutine jump and return management on the other. The on-board MDR (Memory Data Register) is used bidirectionally to buffer memory, but is transparent to the software.

Another XR is the REPEAT register which counts executions at a fixed setting of the Program Counter. Originally, these were to be included in a Program Memory Interface made up of four more 10803s, but we felt that such a use of the 10803 was an extreme overkill for the required functions, the most complex of which is incrementing or decrementing. Instead, we chose to use simple hex counters.

Another XR is the STATUS register which gets most of its inputs (SIGN, ZERO, CARRY, OVERFLOW and LINK) directly from the ALU. It also contains an internal interrupt mask for interrupts based on the status bits. Another status bit is the REPEAT bit which, when set, freezes the program counter and repeats the instruction at that location a number of times based on the contents of the REPEAT register. Three more XRs are used for the generic I/O and are discussed in the I/O subsystem description.

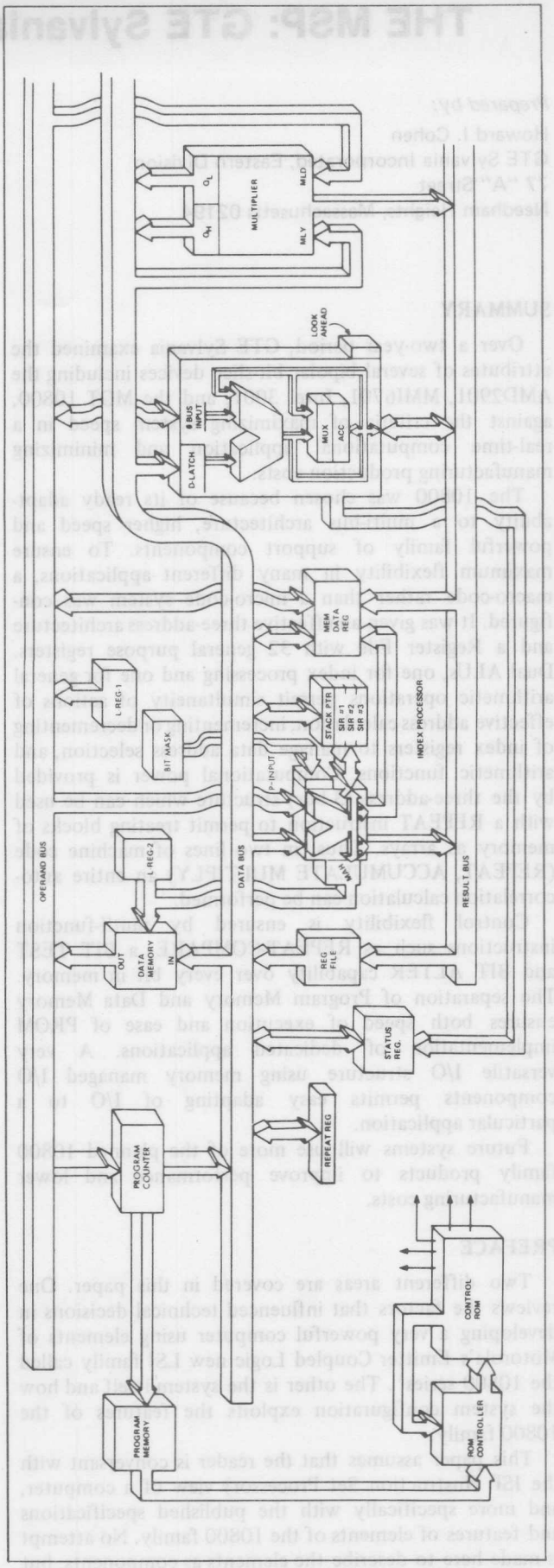


FIGURE 1 – Microsignal Processor

REGISTER FILE

A 32-word by 16-bit general purpose register file sits between the DATA bus which receives its output and the RESULT bus which supplies its input. It serves two explicit functions: functioning in the capacity of up to 32 accumulators, using any of the three SIRs as index registers, or in the capacity of up to 32 different Index Registers for data memory address calculations.

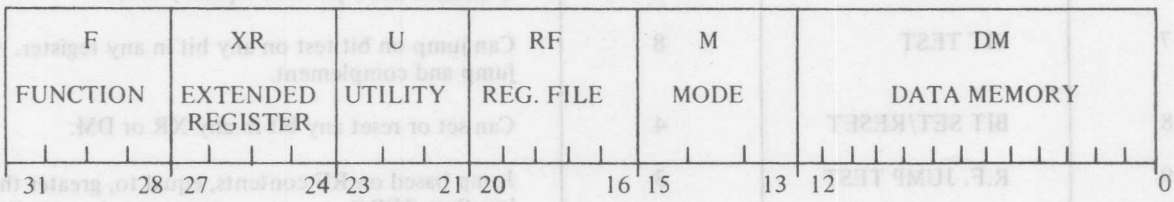
BUS UTILIZATION

The utilization of the buses follows a well-structured pattern. The OPERAND bus gets the principle input from Data Memory. However, if the OPERAND is to be a LITERAL, that is, the OPERAND is contained in the instruction word (see Chart 1), then LREG#1 becomes the input to the OPERAND bus. By also loading LREG#2 with the same field of the instruction word, the address for indirection is taken and immediately cycles the meaningful contents of Data Memory out to the OPERAND bus. If indirect indexing is to be performed, the OPERAND bus value (i.e., the result of the first DM access) is routed into the INDEX PROCESSOR along with the index value which was stored in RF, and summed. The sum is returned to the MAR (Memory Address Register) for actual operand selection in the following cycle. In the meantime, during the same index calculation cycle, the index value on the DATA bus is fed to the ALU via the DATA bus latch. The value is incremented (or decremented) here and returned to the RF via the RESULT bus on the same clock that dropped the calculated Data Memory address into MAR. Thus, any type of Data Memory "moding", i.e., indirection, indexing, etc., takes 100 nanoseconds.

This time becomes the basic micro cycle, encompassing a memory selection taking approximately 35 ns and an ADD cycle taking approximately 55 ns. The remaining 10 ns is used for strobing. Note that the only latching that takes place during this memory access-compute cycle is in the D latch of the 10800 which latches the old output of the RF while the new result is being written into RF In other words, no explicit clocked distinction is made between the memory selection part and the algebraic part of the cycle. The address drives memory whose output drives the bus which drives the ALU. This procedure is also used both in arithmetic instructions performed in the 10800 and multiplies performed in a separate 2-pass 8 by 16 MSI multiplier. The DATA bus is used for RF and XR outputs and is also the return path to Data Memory. The RESULT bus returns arithmetic results to the specified destination. If the destination is Data Memory, routing is through MDR.

Operations involving additions to the RF are very straightforward. The Data Memory contents go to the ALU via the OPERAND bus, while the value in the RF leaves via the DATA bus in the same 100 ns cycle. The result is strobed back into RF via the RESULT bus. Thus, an address calculation, a load, an add and a store are executed in 200 ns or at 5 MOPS. If the operation does not require the address calculation cycle, as in LITERAL + (RF) ⇒ (RF), the operation is performed in 100 ns or at 10 MOPS.

CHART 1 – Instruction Word Format



Where: F-Field is Operation Function
XR-Field is Extended Register
U-Field – Utility or Extension of F-Field
RF – Register File Address
M-Field – Data Memory Selection Mode
DM-Field – Data Memory Address

Note: M concatenated with DM is used for LITERAL operands.

Another example of 3-bus usage is adding the contents of two locations of Data Memory and returning the result to Data Memory. The key to this procedure, which takes 300 ns, is in routing the first operand from the OPERAND bus to be buffered in the MDR. Thus, the first operand is sitting in MDR and being gated out onto the DATA bus via the 10803s data matrix in a pipeline mode. At the same time, the second operand is going out on the OPERAND bus to the ALU. This second cycle ends by latching the result back into MDR via the pipeline feature of that register. The pipelined contents of MDR which are being outputted automatically to the DATA bus can be written into Data Memroy. While these steps are going on, the three Data Memory addresses are being controlled respectively by the three SIRs (Super Index Registers) which are available in the 10803. While each of these three data memory selections are in process, each of the three SIRs are being incremented given the system an ARRAY arithmetic capability.

INSTRUCTION SET

The instruction set was defined to enhance signal analytic computation and software control. The format of the instruction word is illustrated in Chart I. The op code is made up of the F and U field, where F serves as a class or type of instruction and U expands on it to afford up to 8 variations on the possible 16 Fs. In fact, only 11 values of F are used.

XR specifies the extended register address if it enters into the instruction. RF is the register file address, M defines the 8 possible address calculation modes and DM

TABLE 1 – Summary Instruction Description

Function	Name	U-Value	Comment
F=0	MOVE	8	No data change on MOVE.
F=1	ARITH/LOGIC	8	U-defines sources and destination variations (27 = 24) define operation.
F=2	A/L ARRAY	3	Uses SIR's for 3-address control.
F=3	MULTIPLY	8	All products are between XR and DM or LITERAL. Can form fractional, integer or double precision product, and can accumulate sums of products.
F=4	MULT. ARRAY	6	Uses 3-address control via SIR's.
F=5	SHIFT	4	Single place only. Left shift can be circular via LINK bit.
F=6	COMP/SKIP	6	U defines the 6 possible equality tests.
F=7	BIT TEST	8	Can jump on bit test on any bit in any register. Can also jump and complement.
F=8	BIT SET/RESET	4	Can set or reset any bit in any XR or DM.
F=9	R.F. JUMP TEST	3	Jump based on RF contents, equal to, greater than or less than ZERO.
F=A	JUMP/RETURN	6	Includes two absolute JUMPS and two different JSR's both with moves of XR → ACC or ACC → XR. There are two RTN's with the same move options.

addresses up to 8192 words by data memory.

The eleven basic instructions are summarized in Table 1. It should be recognized that each has up to 8 variations based on the assignment of the U field. Where applicable, all effective data memory references are eight combinations of direct or indirect and indexed, with or without increment or decrement. The Register File is set at 32 words, but can be increased by simply making the entire instruction word one bit wider. The same thing can be done with the data memory which is defined as 13 bits.

Setting the REPEAT bit before many of the regular instructions, freezes the program counter while the REPEAT register counts repeated executions of that instruction. The repeating of an ARRAY instruction brings into play the SIRs contained in the 10803s and permits very efficient element-by-element vector operations to be done with only two lines of code. The inclusion of an accumulate multiply combined with a REPEAT makes programming filter calculations or autocorrelations trivially simple. Thus, Σ xiyi is only two lines of machine code. Gain calculations, that is, multiplying an array by a constant is equally efficient. A double precision result can be formed under MULT ARRAY and stored in sequential memory locations.

COMPARE instructions can be repeated with a software-managed internal interrupt occurring on a "hit". The REPEAT register can be monitored through the interrupt subroutine, and the address of the first value to pass the test can be identified. This is used typically to verify the dynamic range of a set of signal samples or test them for limiting. It also speeds up searching operations. Bit testing can be done on any word in any XR, RF or Data Memory. Repeating a bit test of the high order bit is a very simple way to determine when a signal set goes negative. Internal interrupts can be scheduled for any of the bit tests, compare instructions or Condition Codes.

The STACK POINTER Register is one of the registers in the 10803. This permits using data memory for unlimited subroutine stacking. Two words are pushed on an interrupt or JSR and two words are popped on a return: One is the program counter and the other is the STATUS register. The inclusion of the setting of the REPEAT flop in the STATUS register eliminates the problem of interrupting during a REPEAT, and being able to complete it on return. No attempt was made for more context switching in hardware. That task is left to software.

I/O SUBSYSTEM

The generic I/O uses three XRs: The first is called the Direct Data Register and has two parts: an input function and an output function. The MOVE instruction selects the function to be used.

The second XR is called the IDR (Indirect Data Register) and serves as a local control register whose bit assignments are dependent on the application. This permits expanding the DDRs using bits in IDR to indicate which one of several DDRs should be addressed by the one XR value. Bits in the IDR can also be used to define other I/O related protocol requirements such as how

long to time out a delay, management of a UART or USART device, and so on.

The third XR register contains the I/O controls, status and interrupt mask bits. Thus, handshaking or interface sensing can be done either by direct software driving or on an interrupt driven basis.

We have found that the generic technique of three registers is adaptable to the whole gambit from simple to complex interfaces, with minimum hardware problems, and very straightforward software.

FUTURE PLANS

Regarding the other members of the 10800 family, we have not found the 10801 useful because of our macro implementation. We used eight 16 x 4 bit register file chips, but expect to replace them with two 10806s which are 32 by 9 bit two-port devices. The use of two ports will expedite indexing operations. The availability of the 10802 programmable timer will be important in reducing components count through LSI. The planned bidirectional TTL-ECL converters will ease parts count further. The introduction of 4K ECL, RAM and ROM or PROM will ease one of two other implementation problems, namely, memory parts count.

REFERENCES

1. W. Blood, "Motorola's MECL M10800 Family", Motorola, Inc., 1975.

2. H. J. Manley, "GTE Sylvania's PSP: Fast Flexible Signal Processing", Signal, pp. 12-14, January-February 1972.

3. A. J. Goldberg, M. J. Ross, and A. Arcese, "A Predictive Coder for Narrowband Communications Channels in CONF. REC., INT. CONF. COMMUN.", Minneapolis, Minnesota, 1974.

4. A. J. Goldberg, H. L. Shaffer, "A Real-Time Adaptive Predictive Coder Using Small Computers", IEEE Communications Systems and Technology Conference, Dallas, Texas, April 30, 1974.

5. J. deLellis, et al., "A Program Controlled H.F. Modem Implementation", EASCON 74, Washington, D.C., October 7-9, 1974, IEEE Publication 74CHO 883-1AES, pp. 349-352.

M10800 Microprogrammed Demonstrator

Prepared by:
Tom Balph, LSI System Engineer
Ian LeMair, Field Applications Engineer

Motorola I.C. Division
Mesa, Arizona

ABSTRACT

The growth of the LSI bipolar technology within the semiconductor industry has created various microcomputer oriented device families. One of these is Motorola's MECL 10800 group of devices which is a throughput oriented chip set using the MECL 10,000 interface. Based on the MC10800 ALU slice and MC10801 microprogram sequencer, an eight-bit arithmetic processor tied to an M6800 EXORciser system has been designed to exhibit the processing power, microprogram features, and system versatility of bipolar chip sets.

The MECL Microprogrammed On-line Demonstrator (MOD) is a bipolar, microprogrammed unit treated as a peripheral to the M6800 system for data transfer and control. The working system takes advantage of the existing M6800 hardware configuration and software to give the system user easy interface with the MECL processor. Talking through a T.I. 733 terminal, a user can write microcode, store and load microprograms, and run the processor. Features of the MOD include:

1. 8-bit data word
2. 17 scratch pad registers
3. Fully microprogrammed control
4. 1K by 32-bit writable control storage
5. 10 megahertz clock rate
6. Wirewrapped hardware
7. "MODBUG" M6800 operating software

SYSTEM ARCHITECTURE

The block diagram of Figure 1 shows the functional parts and architecture of the system. The MECL MOD unit is assigned peripheral locations on th main M6800 EXORciser bus, and is under control of the M6800 MOS processor. Although the 10800 unit runs independently when activated, the EXORciser acts as master controller to enter and exit data, reset and activate the processor, and load or modify microprogram storage. A T.I. 733 terminal also tied to the system provides keyboard interface, hard copy, and tape cassette storage.

The MOD processor consists of interface, master control, data processing, and microprogram control (including writable control storage) sections interconnected by two main data paths. Data received from the EXORciser bus is transferred to the processor sections via the Incoming Data Bus, and data exits via the Outgoing Data Bus.

INTERFACE

The interface block provides the data buffering and "handshaking" necessary for connecting to the 6800 bus. Based on the 6820 MOS Peripheral Interface Adapter (PIA) LSI device, the MOD interface occupies several addresses on the 6800 memory map. Contained within these addresses are registers for control of the PIA, the master control register address, read and write addresses for data processor, and addresses for microprogram control. The interface logic decodes these locations and sets up timing and data buffering used for information exchange within the system.

The two MOD eight-bit data busses are tied to the interface section. All data exchange moves through these busses to and from the ECL logic.

MASTER CONTROL

The master control register along with its decode and timing logic oversees operation of the processor. The register contains a seven-bit code which determines the operating mode of the ECL unit. Its functions include:

1. Reset the processor.
2. Activate the processor.
3. Load WCS word address.
4. Load WCS page address.
5. Load WCS data.
6. Read WCS data.

The master control register occupies a separate 6800 peripheral address and is written to only. On receiving a given code, one of the above functions or a combination of functions is decoded and implemented. In this manner, the user can load and read WCS, and can reset or activate the processor. When the processor is activated, the master register relinquishes control to the stored microprogram. However, control is returned to the master register anytime a non-activate code is stored.

The decode logic generates the proper control timing signals for the selected function. Timing synchronization is necessary because the EXORciser runs from a 1 MHz, 2 phase clock and the MOD processor uses a free-running 10 MHz single phase clock. Timing signals from the interface circuitry along with the control code in the register are synchronized to the 10 MHz master oscillator, and, in turn, controls timing in the rest of the processor.

DATA PROCESSING

The ALU, working registers, and condition code register are contained in the data processing section (Figure 2). The eight-bit ALU uses two MC10800 four-bit ALU slices (Figure 3) which also contains the accumulator. The register file contains sixteen (16) locations and uses MCM10145 ECL RAM's.

The data processing section bus structure is designed to take advantage of the 10800 features and minimize package count. Data entering via the Incoming Data Bus must be transferred through the ALU for loading into any register. The accumulator, register file, and condition code register are all loaded directly from the Outgoing Data Bus. The register file and C.C. register in turn drive the File Bus to ALU.

The data ports of the 10800 are connected to the busses as shown in Figure 2. The File Bus is connected to the Ø port of the 10800 taking advantage of the Ø bus latch. The latch allows the RAM based register file to function as a master-slave design which makes R.F. read/write possible in one clock cycle. The Incoming

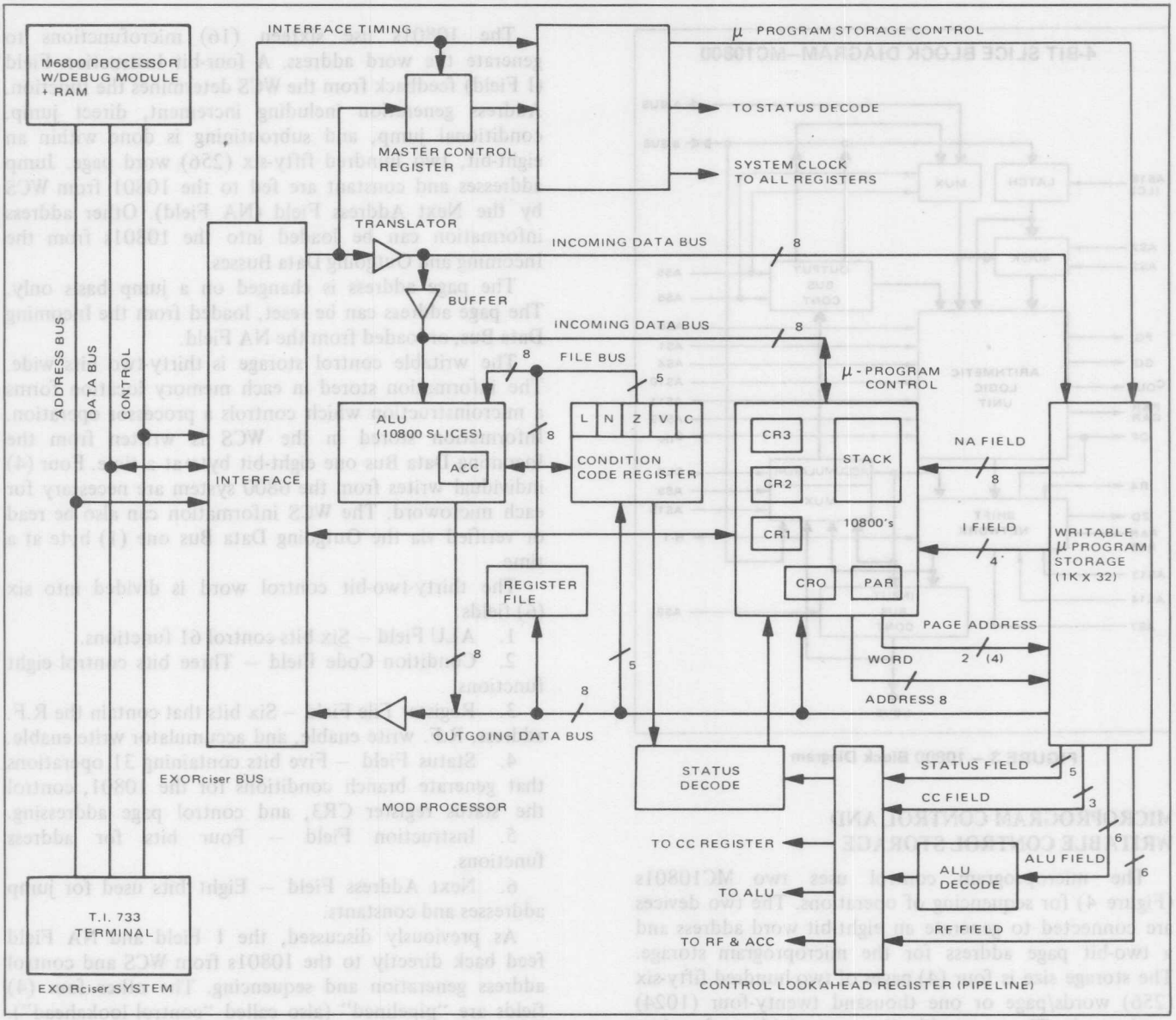


FIGURE 1 – System Architecture

DATA PROCESSING SECTION

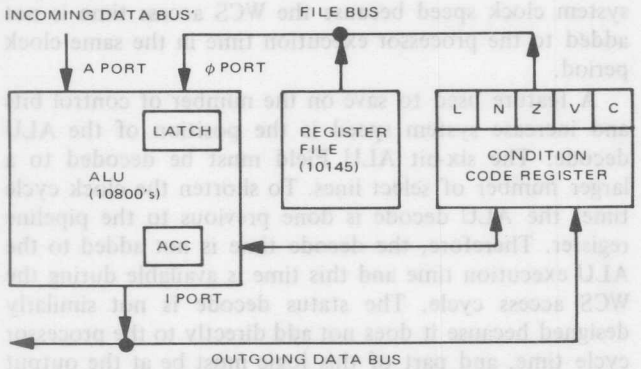


FIGURE 2 – Data Processing Section

Data Bus is connected to the A port of the 10800, and the ALU/shifter results appear at the I port which drives the Outgoing Data Bus.

The ALU function set uses the accumulator, register file (File Bus), and Incoming Data Bus as operand sources. The functions include transfer, logic, shift, and arithmetic operations. Special Add/Subtract-shift and BCD arithmetic functions are useful for "number crunching" operations and display special features of the 10800.

The R.F., accumulator, and C.C. register are controlled independently of the ALU. The R.F. and accumulator have a separate control field and can be written on independently. The C.C. register also has its own control field.

The condition code register contains five bits which are:

1. C – carry out from the most signficant bit of the ALU.
2. V – two's complement arithmetic overflow.
3. Z – zero detect of the result.
4. N – sign of the result.
5. L – link bit for shift.

The register can be loaded from the Outgoing Data Bus or from the ALU. The individual bits can be loaded from the ALU as needed for logic, arithmetic, and shift operations.

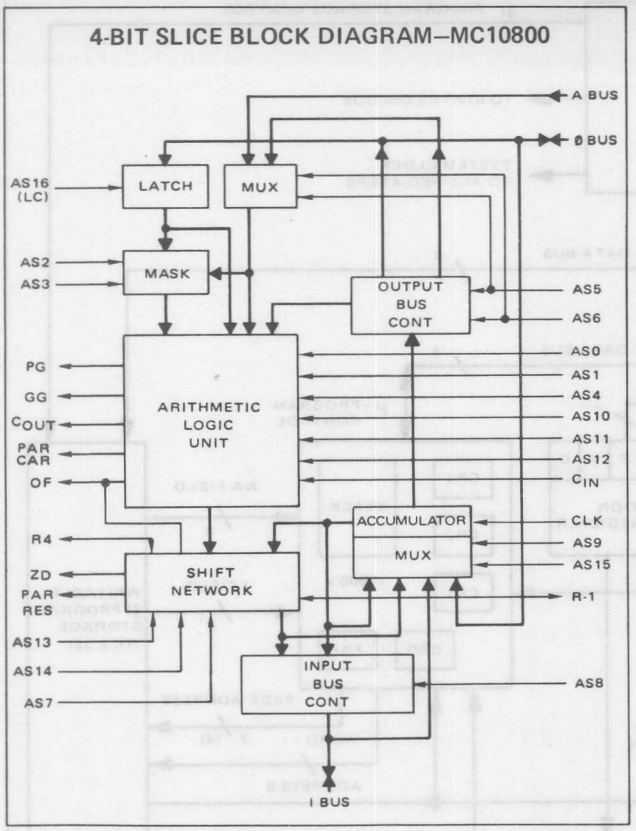


FIGURE 3 – 10800 Block Diagram

MICROPROGRAM CONTROL AND WRITABLE CONTROL STORAGE

The microprogram control uses two MC10801s (Figure 4) for sequencing of operations. The two devices are connected to generate an eight-bit word address and a two-bit page address for the microprogram storage. The storage size is four (4) pages of two hundred fifty-six (256) words/page or one thousand twenty-four (1024) total words. The page address is expandable to four bits for up to sixteen (16) pages (4K total words).

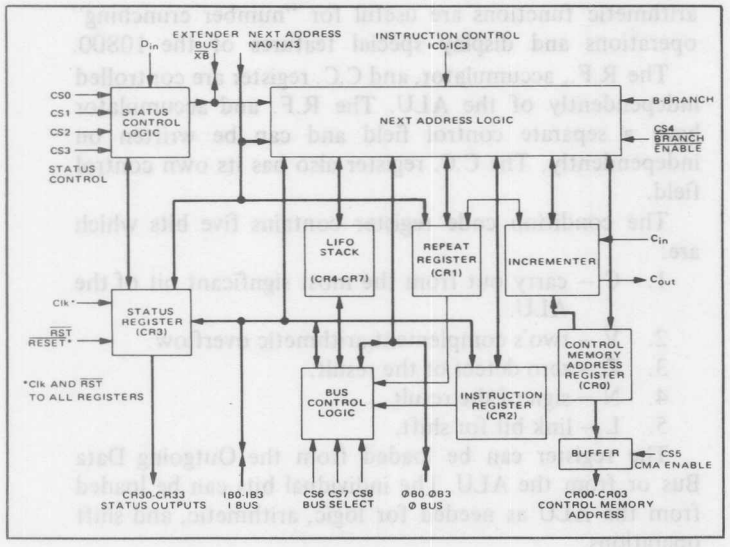


FIGURE 4 – 10801 Block Diagram

The 10801s use sixteen (16) microfunctions to generate the word address. A four-bit Instruction Field (I Field) feedback from the WCS determines the function. Address generation including increment, direct jump, conditional jump, and subroutining is done within an eight-bit, two hundred fifty-six (256) word page. Jump addresses and constant are fed to the 10801 from WCS by the Next Address Field (NA Field). Other address information can be loaded into the 10801s from the Incoming and Outgoing Data Busses.

The page address is changed on a jump basis only. The page address can be reset, loaded from the Incoming Data Bus, or loaded from the NA Field.

The writable control storage is thirty-two bits wide. The information stored in each memory location forms a microinstruction which controls a processor operation. Information stored in the WCS is written from the Incoming Data Bus one eight-bit byte at a time. Four (4) individual writes from the 6800 system are necessary for each microword. The WCS information can also be read or verified via the Outgoing Data Bus one (1) byte at a time.

The thirty-two-bit control word is divided into six (6) fields:

1. ALU Field – Six bits control 61 functions.
2. Condition Code Field – Three bits control eight functions.
3. Register File Field – Six bits that contain the R.F. address, R.F. write enable, and accumulator write enable.
4. Status Field – Five bits containing 31 operations that generate branch conditions for the 10801, control the status register CR3, and control page addressing.
5. Instruction Field – Four bits for address functions.
6. Next Address Field – Eight bits used for jump addresses and constants.

As previously discussed, the I Field and NA Field feed back directly to the 10801s from WCS and control address generation and sequencing. The other four (4) fields are "pipelined" (also called "control lookahead"). The pipelined fields are buffered by a register and these operations occur one clock cycle later than the I Field and NA Field control operation. This feature allows a higher system clock speed because the WCS access time is not added to the processor execution time in the same clock period.

A feature used to save on the number of control bits and increase system speed is the position of the ALU decode. The six-bit ALU Field must be decoded to a larger number of select lines. To shorten the clock cycle time, the ALU decode is done previous to the pipeline register. Therefore, the decode time is not added to the ALU execution time and this time is available during the WCS access cycle. The status decode is not similarly designed because it does not add directly to the processor cycle time, and part of this logic must be at the output of the pipeline register.

PROGRAMMING MODEL AND MICROINSTRUCTION SET

The microinstruction set makes a large number of registers available to the user. The accumulator and register file are controlled from the RF Field, the condi-

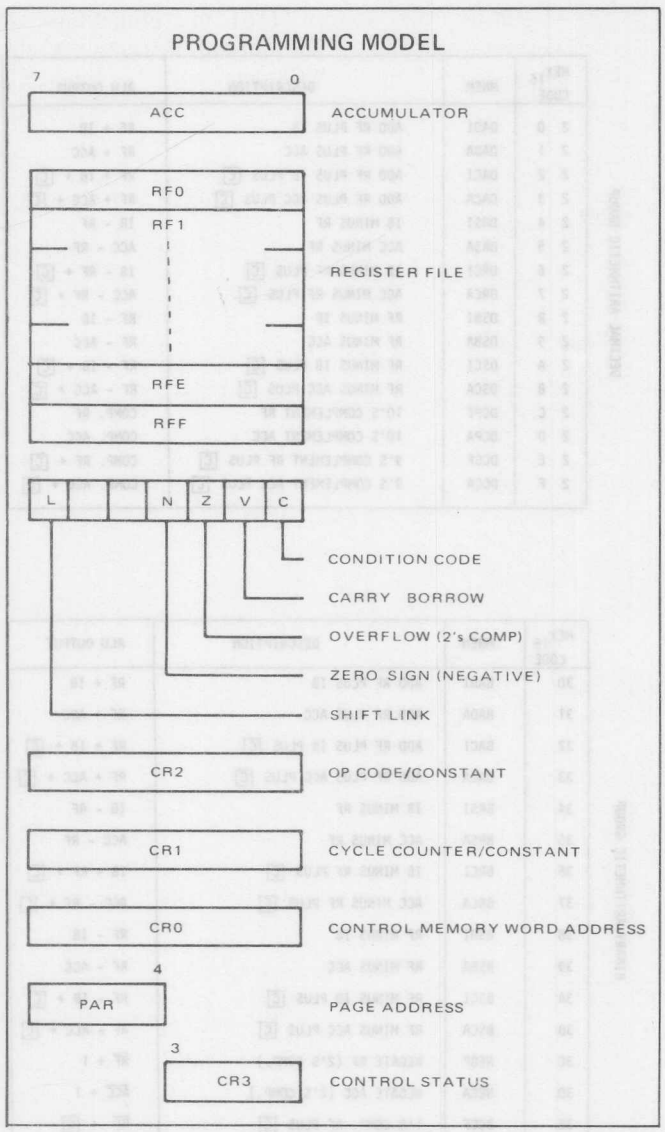


FIGURE 5 – Programming Model

tion code register is controlled from the C.C. Field, and the remaining registers (Figure 5) are manipulated by the status, instruction, and NA Fields. The microinstruction set is listed in Figure 6.

PERFORMANCE EXAMPLE

Several types of problems have been programmed on the MOD. A good example of this is a full floating point single precision multiply program. The format for numbers is as follows:

(8-bit sign and exponent)
times (1)
(24-bit mantissa)

Each number occupies four (4) eight-bit bytes. The program produces:

(multiplicand) x (multiplier)
equals (2)
product

Worst case multiply time is 37.6 microseconds.

HARDWARE

The 10800 MOD processor consists of three (3) wire-wrap boards that plug into the EXORciser chassis. Total device count consists of about 80 MSI/SSI devices (TTL, ECL, and Linear Interface), four (4) 10800 ECL/LSI devices, a 6820 MOS PIA, and thirty-two (32) ECL 1K RAMS (10146). Power supplies consist of +5.0 volts and +3.0 volts. Clock frequency is 10 MHz.

SOFTWARE

"MODBUG" support software is written in M6800 assembler language and consists of:

1. System initialization and writable control store monitor. This allows active interface of writing and reading of control storage.
2. A punch or memory dump capability to store the control memory contents on magnetic tape.
3. A memory load capability from magnetic tapes.
4. An Activate/Interface monitor to control and to load and receive data from the processor when running.

SUMMARY OF PURPOSE AND CAPABILITY

The main goal of the 10800 MOD processor is to provide a demonstration vehicle for microprogramming, ECL LSI, and the total software and hardware interface of a working system. The arithmetic capability is designed to be a "number cruncher" performing multiply, divide, floating point, and higher level functions with full BCD featured.

ALU FIELD LISTING

TRANSFER GROUP

HEX CODE[16]	MNEM	DESCRIPTION	ALU OUTPUT
0 0	TIB	TRANSFER INCOMING DATA BUS	I B
0 1	TAC	TRANSFER ACCUMULATOR	ACC
0 2	TRF	TRANSFER REGISTER FILE	R F
0 3	IAC	INVERT ACCUMULATOR	$\overline{ACC}$
0 4	IRF	INVERT REGISTER FILE	$\overline{R\ F}$
0 5	-	-	-
0 6	-	-	-
0 7	DECF	DECREMENT REGISTER FILE	RF-1

LOGIC GROUP

HEX CODE[16]	MNEM	DESCRIPTION	ALU OUTPUT
0 8	ANDI	I BUS AND REGISTER FILE	IB ∩ RF
0 9	ANDA	ACC AND REGISTER FILE	ACC ∩ RF
0 A	ZERO	ALL ZERO	0'S
0 B	ONES	ALL ONES	1'S
0 C	EXOI	I BUS EXCLUSIVE-OR REGISTER FILE	IB ⊕ RF
0 D	EXOA	ACC EXCLUSIVE-OR REGISTER FILE	ACC ⊕ RF
0 E	IORI	I BUS INCLUSIVE-OR REGISTER FILE	IB U RF
0 F	IORA	ACC INCLUSIVE-OR REGISTER FILE	ACC U RF

SHIFT GROUP

HEX CODE[16]	MNEM	DESCRIPTION	ALU OUTPUT
1 0	ASRF	ARITHMETIC SHIFT RIGHT R.F.	→ RF → L
1 1	ASRA	ARITHMETIC SHIFT RIGHT ACC	→ ACC → L
1 2	SASR	ARITH. SHIFT RIGHT (ACC MINUS RF)	→ ACC-RF → L
1 3	AASR	ARITH. SHIFT RIGHT (ACC PLUS RF)	→ ACC+RF → L
1 4	LSRF	LOGIC SHIFT RIGHT RF	0 → RF → L
1 5	LSRA	LOGIC SHIFT RIGHT ACC	0 → ACC → L
1 6	RORF	ROTATE RIGHT RF	→ RF → L
1 7	RORA	ROTATE RIGHT ACC	→ ACC → L
1 8	ASLF	ARITHMETIC SHIFT LEFT RF	L ← RF ← 0
1 9	ASLA	ARITHMETIC SHIFT LEFT ACC	L ← ACC ← 0
1 A	ROLF	ROTATE LEFT RF	L ← RF ←
1 B	ROLA	ROTATE LEFT ACC	L ← ACC ←
1 C	ADSL	SHIFT LEFT (ACC PLUS RF)	L ← ACC+RF ←
1 D	SUSL	SHIFT LEFT (ACC MINUS RF)	L ← ACC-RF ←
1 E	ACSR	SHIFT RIGHT W/CARRY (ACC PLUS RF)	C → ACC+RF → L
1 F	-	-	

DECIMAL ARITHMETIC GROUP

HEX$_{16}$ CODE	MNEM	DESCRIPTION	ALU OUTPUT
2 0	DADI	ADD RF PLUS IB	RF + IB
2 1	DADA	ADD RF PLUS ACC	RF + ACC
2 2	DACI	ADD RF PLUS IB PLUS [C]	RF + IB + [C]
2 3	DACA	ADD RF PLUS ACC PLUS [C]	RF + ACC + [C]
2 4	DRSI	IB MINUS RF	IB - RF
2 5	DRSA	ACC MINUS RF	ACC - RF
2 6	DRCI	IB MINUS RF PLUS [C]	IB - RF + [C]
2 7	DRCA	ACC MINUS RF PLUS [C]	ACC - RF + [C]
2 8	DSBI	RF MINUS IB	RF - IB
2 9	DSBA	RF MINUS ACC	RF - ACC
2 A	DSCI	RF MINUS IB PLUS [C]	RF - IB + [C]
2 B	DSCA	RF MINUS ACC PLUS [C]	RF - ACC + [C]
2 C	DCPF	10'S COMPLEMENT RF	COMP. RF
2 D	DCPA	10'S COMPLEMENT ACC	COMP. ACC
2 E	DCCF	9'S COMPLEMENT RF PLUS [C]	COMP. RF + [C]
2 F	DCCA	9'S COMPLEMENT ACC PLUS [C]	COMP. ACC + [C]

BINARY ARITHMETIC GROUP

HEX$_{16}$ CODE	MNEM	DESCRIPTION	ALU OUTPUT
30	BADI	ADD RF PLUS IB	RF + IB
31	BADA	ADD RF PLUS ACC	RF + ACC
32	BACI	ADD RF PLUS IB PLUS [C]	RF + IB + [C]
33	BACA	ADD RF PLUS ACC PLUS [C]	RF + ACC + [C]
34	BRSI	IB MINUS RF	IB - RF
35	BRSA	ACC MINUS RF	ACC - RF
36	BRCI	IB MINUS RF PLUS [C]	IB - RF + [C]
37	BRCA	ACC MINUS RF PLUS [C]	ACC - RF + [C]
38	BSBI	RF MINUS IB	RF - IB
39	BSBA	RF MINUS ACC	RF - ACC
3A	BSCI	RF MINUS IB PLUS [C]	RF - IB + [C]
3B	BSCA	RF MINUS ACC PLUS [C]	RF - ACC + [C]
3C	NEGF	NEGATE RF (2'S COMP.)	$\overline{RF}$ + 1
3D	NEGA	NEGATE ACC (2'S COMP.)	$\overline{ACC}$ + 1
3E	BCCF	1'S COMP. RF PLUS [C]	$\overline{RF}$ + [C]
3F	BCCA	1'S COMP. ACC PLUS [C]	$\overline{ACC}$ + [C]

CONDITION CODE FIELD LISTING

HEX$_{16}$ CODE	MNEM	DESCRIPTION	C.C. REG. L	N	Z	V	C
0	RTL	RESET LINK (0 → L)	R	·	·	·	·
1	LDA	LOAD ALL BITS FROM ALU	↕	↕	↕	↕	↕
2	LDB	LOAD OUTPUT DATA BUS	↕	↕	↕	↕	↕
3	IHB	INHIBIT ALL	·	·	·	·	·
4	STL	SET LINK (1 → L)	S	·	·	·	·
5	LAR	LOAD ARITHMETIC (Z,N,C,Y)	·	↕	↕	↕	↕
6	LOG	LOAD LOGIC (Z,N)	·	↕	↕	·	·
7	ENB	ENABLE TO FILE BUS	·	·	·	·	·

· = NO CHANGE
↕ = LOAD DATA
R = RESET
S = SET

REGISTER FILE FIELD LISTING

RF ADDRESS

	REGISTER
0	RF0
1	RF1
2	RF2
3	RF3
4	RF4
5	RF5
6	RF6
7	RF7
8	RF8
9	RF9
A	RFA
B	RFB
C	RFC
D	RFD
E	RFE
F	RFF

RF WRITE ENABLE

	DESCRIPTION
0	NO CHANGE
1	WRITE Ø BUS INTO RF

ACC WRITE ENABLE

	DESCRIPTION
0	NO CHANGE
1	WRITE Ø BUS INTO ACC

FIGURE 6 – Microinstruction Set

STATUS FIELD LISTING

HEX_{16} CODE	NMEM	DESCRIPTION
00	TSLS	TEST OUTGOING DATA BUS LSB
01	TSTN	TEST SIGN BIT [N]
02	TSTZ	TEST ZERO DETECT [Z]
03	TSTV	TEST OVERFLOW [V]
04	TSTC	TEST CARRY BIT [C]
05	TSTL	TEST LINK BIT [L]
06	TSMS	TEST OUTGOING DATA BUS MSB
07	TONE	TEST CONDITION = 1
08	TSLL	TEST OUTGOING DATA BUS LSB & [L]
09	TSNZ	TEST SIGN [N] & ZERO DETECT [Z]
0A	TRR1	TRANSFER CR1 TO INCOMING DATA BUS
0B	TRR2	TRANSFER CR2 TO INCOMING DATA BUS
0C	RTSR	RESET STATUS REGISTER CR3
0D	LIBS	LOAD INCOMING DATA BUS INTO CR3
0E	LNAS	LOAD N.A. FIELD INTO CR3
0F	-	-
10	LDB0	LOAD D_{IN} INTO $CR3_0$
11	LDB1	LOAD D_{IN} INTO $CR3_1$
12	LDB2	LOAD D_{IN} INTO $CR3_2$
13	LDB3	LOAD D_{IN} INTO $CR3_3$
14	TSB0	TEST $CR3_0$
15	TSB1	TEST $CR3_1$
16	TSB2	TEST $CR3_2$
17	TSB3	TEST $CR3_3$
18	STB0	SET $CR3_0$
19	STB1	SET $CR3_1$
1A	STB2	SET $CR3_2$
1B	STB3	SET $CR3_3$
1C	RTUA	RESET PAGE (UPPER) ADDRESS
1D	LIBU	LOAD I. BUS INTO PAGE ADDRESS
1E	LNAU	LOAD N.A. INTO PAGE ADDRESS
1F	INHS	INHIBIT STATUS (NOP)

I FIELD LISTING

TABLE I

							REGISTER AND FLIP-FLOP OUTPUTS V_{OL} ⟷ V_{OH}				
MNEM	CODE IC3	IC2	IC1	IC0	DESCRIPTION	BRANCH OR REPEAT CONDITION[2]	CR0	CR1	CR2	LIFO STACK CR4–CR7	RSQ[3]
INC	1	1	0	0	INCREMENT	X	CR0 plus 1	–	–	–	–
JMP	0	0	1	0	JUMP TO NEXT ADDRESS	X	NA	–	–	–	–
JIB	1	0	0	0	JUMP TO I BUS	X	IB·NA	–	–	–	–
JIN	1	0	0	1	JUMP TO I BUS & LOAD CR2	X	IB·NA	–	IB	–	–
JPI	1	0	1	0	JUMP TO PRIMARY INST.	X	CR2·NA	–	–	–	–
JEP	1	1	1	0	JUMP TO EXTERNAL PORT	X	OB·NA	–	–	–	–
JL2	0	0	0	1	JUMP & LOAD CR2	X	NA	–	IB	–	–
JLA	0	0	1	1	JUMP & LOAD ADDRESS	X	NA	CR0 plus 1	–	–	–
JSR	0	0	0	0	JUMP TO SUBROUTINE	RSQ=1 and CR1≠FF	NA	–	–	"PUSH" CR0 TO STACK	–
						RSQ=0 or CR1 = FF	NA	–	–	"PUSH TO CR0+1 TO STACK	–
RTN	1	1	1	1	RETURN FROM SUBROUTINE	RSQ=1 and CR1≠FF	CR4	CR1 plus 1	–	"POP" STACK TO CR0	–
						RSQ=0 or CR1=FF	CR4	–	–	"POP" STACK TO CR0	0
RSR	1	1	0	1	REPEAT SUBROUTINE	X	CR0 plus 1	NA	–	–	1
RPI	1	0	1	1	REPEAT INSTRUCTION	RSQ=1 and CR1≠FF	–	CR1 plus 1	–	–	–
						RSQ=0 or CR1=FF	CR1·NA	–	–	–	0
BRC	0	1	0	1	BRANCH ON CONDITION	1	NA	–	–	–	–
						0	CR0 plus 1	–	–	–	–
BSR	0	1	0	0	BRANCH TO SUBROUTINE	1	NA	–	–	"PUSH" CR0+1 TO STACK	–
						0	CR0 plus 1	–	–	–	–
ROC	0	1	1	1	RETURN ON CONDITION	1	CR4	–	–	"POP" STACK TO CR0	0
						0	NA	–	–	–	–
						UPPER	NA	–	–	–	–
BRM	0	1	1	0	BRANCH & MODIFY	LOWER	$CR0_0$=NA0·B0 $CR0_1$=NA1·B1 $CR0_2$=NA2 $CR0_3$=NA3	–	–	–	–

NOTES: 1. X = DON'T CARE
– = NO CHANGE
2. BRANCH CONDITION SELECTED BY STATUS FIELD.
3. RSQ = STATE OF REPEAT FLIP-FLOP.

FIGURE 6 – Microinstruction Set (continued)